Billboards

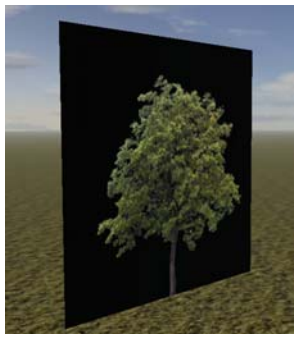
- Replace a mesh with a pre-computed picture of the mesh
- Fast to render: a single textured polygon versus many
- Sometimes called “imposters”

What to use it for?

- Scenery
  - Trees, grass, spectators
- Mesh simplification
  - Replace far-away objects with billboards
- Non-polygonal objects
  - Fire, smoke, clouds, particles

Billboard basics

- A billboard is a textured rectangle
- Texture is static
- Draw billboard where mesh would have been drawn



Billboard basics

- Need to remove texture background
- Two ways:
  - Masking
  - Alpha blending

Billboards using Masking

- Each texel (texture element) has alpha value 0 or 1
- Only draw texels with alpha value 1

```
void glAlphaFunc(GLenum func, GLclampf ref);
     func: when to render the texel
           GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER,
           GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS
     ref: value to compare texel alpha with

glAlphaFunc(GL_GREATER, 0.5f);
glEnable(GL_ALPHA_TEST);
```

Billboards using Masking (cont.)

- Advantages:
  - Fast to render
  - Always works, independent of drawing order
- Disadvantages:
  - Hard edges
  - Making mask from real image difficult

Billboards using Blending

- Each texel has an alpha value in the range from 0 to 1
- Alpha value is used to blend each texel with the background

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
     sfactor: source (incoming pixel) blending factor,
              lots of different modes, check OpenGL docs
     dfactor: destination (existing pixel) blending factor,
              lots of different modes, check OpenGL docs
```

  - Generally, `RGBA = RGBA_s * sfactor + RGBA_d * dfactor`
  - where the factors indicate constants, or one or more elements of `RGBA_s` or `RGBA_d`

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
// (R,G,B,A) = (R_s,G_s,B_s,A_s) * A_s + (R_d,G_d,B_d,A_d) * (1-A_s)
glEnable(GL_BLEND);
```

- Must turn off depth buffer culling (unless always drawing back to front): `glDisable(GL_DEPTH_TEST);`

Billboards using Blending (cont.)

- Advantages:
  - Smooth blending with background
  - Partially transparent objects possible
- Disadvantages:
  - Takes longer to render
  - Rendering must be done in depth sorted order (front-to-back or back-to-front)

Initial: $RGBA_{12} = 0$

Blend source 1: $RGBA'_{12} = RGBA_{S1} * A_{S1} + RGBA_{12} * (1-A_{S1}) = RGBA_{S1} * A_{S1}$

Blend source 2: $RGBA''_{12} = RGBA_{S2} * A_{S2} + RGBA'_{12} * (1-A_{S2}) = RGBA_{S2} * A_{S2} + RGBA_{S1} * A_{S1} * (1-A_{S2})$
$= RGBA_{S2} * A_{S2} + RGBA_{S1} * A_{S1} - RGBA_{S1} * A_{S1} * A_{S2}$

Reverse:

Initial: $RGBA_{21} = 0$

Blend source 2: $RGBA'_{21} = RGBA_{S2} * A_{S2} + RGBA_{21} * (1-A_{S2}) = RGBA_{S2} * A_{S2}$

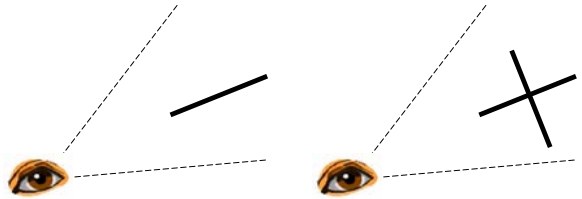
Blend source 1: $RGBA''_{21} = RGBA_{S1} * A_{S1} + RGBA'_{21} * (1-A_{S1}) = RGBA_{S1} * A_{S1} + RGBA_{S2} * A_{S2} * (1-A_{S1})$
$= RGBA_{S1} * A_{S1} + RGBA_{S2} * A_{S2} - RGBA_{S2} * A_{S2} * A_{S1}$
$!= RGBA''_{12}$

  - Blending is generally render order dependent
  - Some blending method are order independent, e.g. additive blending:

```
glBlendFunc(GL_ONE, GL_ONE);
// ((0) * 1 + RGBA_1 * 1) * 1 + RGBA_2 * 1 = ((0) * 1 + RGBA_2 * 1) * 1 + RGBA_1 * 1

glBlendFunc(GL_SRC_ALPHA, GL_ONE);
// ((0) * 1 + RGBA_1 * A_1) * 1 + RGBA_2 * A_2 = ((0) * 1 + RGBA_2 * A_2) * 1 + RGBA_1 * A_1
```

Multiple billboards

- One billboard makes a very flat tree from some angles
- Use two in a cross configuration



One tree makes a forest

- Drawing one billboard is cheap
- Can draw many billboards using the same texture
- Add variation by randomly:
  - scaling the billboards
  - rotating around the symmetry axis
  - changing overall shading for each billboard

One tree makes a forest (cont.)

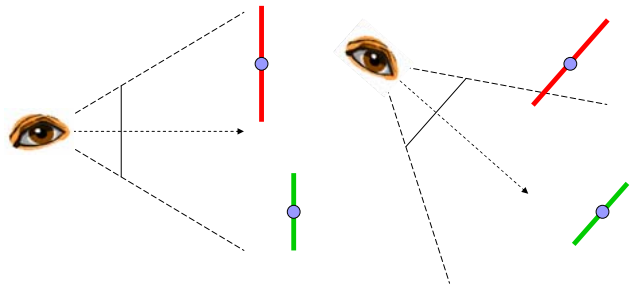
```
// Draw each tree
for(vector<Tree>::const_iterator ti = forest.begin(); ti != forest.end(); ti++)
{
    glPushMatrix();

    // Place the tree
    glTranslatef(ti->x, 0, ti->z);
    // Scale the tree
    glScalef(ti->widthscale, ti->heightscale, ti->widthscale);
    // With the GL_MODULATE texture mode the texture colours
    // are multiplied by the glColor values. This is an easy
    // way to make a texture lighter or darker, or even give
    // it a tint.
    glColor3f(ti->lightness, ti->lightness, ti->lightness);
    glRotatef(ti->rot, 0,1,0);

    glBegin(GL_QUADS);
          glTexCoord2f(0, 1); glVertex3f(-1, 0, 0);
          glTexCoord2f(0, 0); glVertex3f(-1, 2, 0);
          glTexCoord2f(1, 0); glVertex3f( 1, 2, 0);
          glTexCoord2f(1, 1); glVertex3f( 1, 0, 0);
    glEnd();

    glPopMatrix();
}
```

Aligning billboards

- Billboard polygons obvious when viewing nearly edge-on
- Possible solution: lock billboard rotation axes to camera rotation
  - Billboard always facing camera
  - Need only one billboard per object
  - Works best for symmetric objects in a crowded scene (trees in forest, people in a stadium)
  - Often only lock heading, not pitch/bank

Align with projection plane

- Set billboard rotation such that it faces the camera's projection plane



Align with projection plane

- Rotations around locked axes are zero relative to camera rotation
- Two ways of implementing:
  1. Set billboard rotation of locked axes equal to camera rotation
  2. Render billboards in camera coordinates without rotation

Set to camera rotation

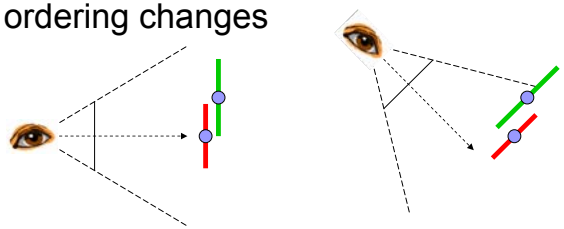
- Generally know camera rotations around each axis from game state
- Set rotation around each locked axis equal to the matching camera rotation
  - Easy to implement
  - Can lock 1, 2, or 3 rotation axes
  - Extra cost of setting rotation for each billboard

Render in camera coords

- Set modelview matrix to identity
- Compute position of billboard in camera coordinate system
- Render billboard at position without any rotation
  - No need to set rotation for each billboard
  - Locks all rotation axes
  - Need to compute camera-relative positions

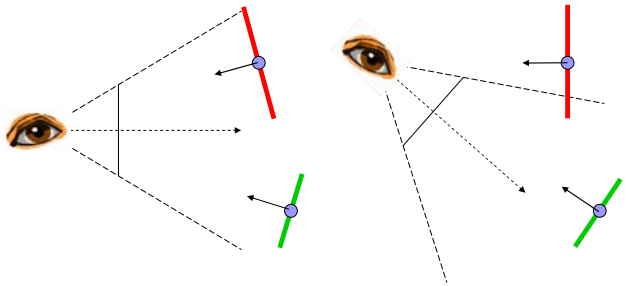
Align with projection plane

- Billboard rotation remains steady with linear movement
- "Popping" while rotating as billboard ordering changes



Align with camera position

- Billboard rotated so that normal points at camera position



Align with camera position

- Compute vector from billboard to camera
- Heading/pitch of vector gives rotation for locked heading/pitch
- Can use trigonometry
  - Easy when locking only heading, ok with pitch
  - Inverse trig functions are expensive
- Use vector algebra for general answer
  - Construct coord system for billboard using camera to centre of billboard vector, and an "up" vector
  - Put in matrix and multiply with modelview matrix

Align with camera position

$C = Pos_{BB} - Pos_{Cam}$

$Up = (0,1,0)$

$X_{BB} = (Up \times C) / |Up \times C|$

$Y_{BB} = (C \times X_{BB}) / |C \times X_{BB}|$

$Z_{BB} = C / |C|$

$$M_{BB} = \begin{pmatrix} Xx & Yx & Zx & 0 \\ Xy & Yy & Zy & 0 \\ Xz & Yz & Zz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`glMultMatrix(`M_{BB}`);`

- Keeps billboard pointing upwards
- Note that this will fail when `C` is colinear with `Up`
  - Camera looking straight down at billboard, so don't render
- Can orient the billboard any way you like by choosing an appropriate "up" vector (e.g. point it in the same direction as camera up)

Align with camera position

- Eliminates almost all popping
  - Only nearly co-planar billboards may pop
- Billboards steady with camera rotation
- Billboards rotate with linear movement
  - Do not use wide field of view
  - Side-effect: avoids camera going through billboard

Finishing touches: shadows

- Simple shadows on flat ground
  - Render rectangle on ground using mask of tree as texture
  - Mask is the tree texture
    - RGB set to 0
    - Alpha set to 0 for texels of the tree, and 1 for texels of the background
    - Possibly have Alpha go smoothly from 0 to 1 around the edges of the tree for softer shadows
  - Shadow rectangle blended with terrain using

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

  - As shadows always appear on top of terrain, turn off depth testing, else may get "Z-fighting" as the shadow rectangles and terrain are co-planar
    - Z-fighting happens when polygons are co-planar and tiny floating point errors makes the depth testing pass or fail randomly for different pixels
    - `glDisable(GL_DEPTH_TEST)`
  - Shadows rendered before trees are rendered

Finishing touches: skybox

- Draw sky as an inward-facing cube
- Texture each side with a view of a sky
- Texture only correct for the point of view from which it was generated
- Keep the skybox centred on the camera
  - The sky is very (infinitely) far away
  - Camera never gets closer to the sky
  - Sky texture always appears correct
- Skybox drawn before anything else
  - Turn off depth buffer writing `glDepthMask(GL_FALSE)`
  - Draw unit-sized cube around camera
    - Sky looks the same no matter what size the cube is
- Use fog to hide the limited size of the terrain and make a more natural transition to sky
  - Skybox not affected by fog, as it is actually drawn very near the camera
  - Alternatively, turn off fog for the skybox, or else skybox will by nothing but fog
- Note that OpenGL1.3 has a special cube-mapping texture mode for texturing a skybox using one texture, instead of needing one for each side

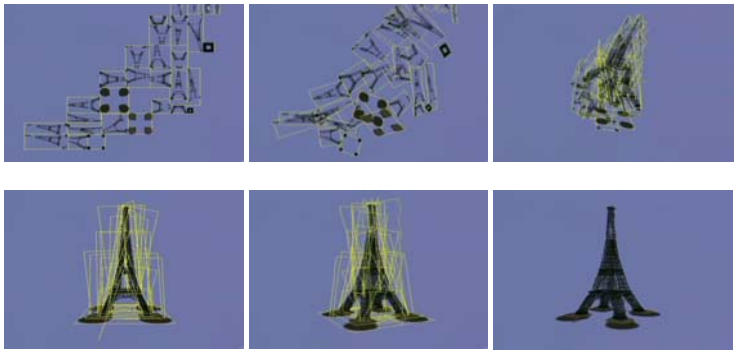
Billboard Clouds

- "Billboard Clouds for Extreme Model Simplification" – Décoret et al., SIGGRAPH 2003
- Use 10-100 billboards to approximate a mesh
- Need to find set of planes which best fit a mesh
  - Offline error minimisation process ~1 minute

Billboard Clouds

- Create texture for each billboard by projecting matching polygons
  - Texture size can be chosen based on billboard size
- Render all billboards
  - Billboards rotate and move with object
- Use fewer billboards for far-away views, more for nearby views

Billboard Clouds



SpeedTree http://www.idvinc.com/speedtree

- Commercial library for rendering trees, grass, plants fast
- Complex scenes rendered quickly by using billboards:
  - Tree far away rendered as single billboard
  - Nearby tree rendered using polygon trunk and branches, with billboards for each bunch of leaves
  - Visually smooth transition between the two